**Step 0: Data Preprocessing**

* **Objective**: Combine 19 project-specific datasets into a unified knowledge base, preprocess text for embeddings, and store filter fields as metadata.
* **Action**: Merge datasets, clean and normalize text, and prepare metadata for post-search filtering.
* **Details**:
  + Load 19 Dataiku datasets
  + Drop rows with missing observation or solution.
  + Clean text columns (observation, solution)
  + Fill missing metadata (project, database, language) with mode values per dataset.
  + Standardize language to ISO codes (e.g., en, fr, it, kk, ru, es, sv).
  + Keep metadata: project, fleet, subsystem, database, observationcategory, problemcode, problemcause, solutioncategory, language, failureclass, date.
  + Save to combined\_data.csv and Dataiku dataset combined\_data.
* **Outcome**: A clean, unified dataset for your smaller version, ready for embedding generation.

**Step 1: Select a Multilingual LLM**

* **Objective**: Identify a model for cross-lingual understanding across 7 languages (scalable to 11).
* **Action**: Load paraphrase-multilingual-mpnet-base-v2 via sentence-transformers, validating support for English, French, Italian, Kazakh, Russian, Spanish, Swedish.
* **Details**:
  + Test with sample sentences in all 7 languages.
  + Confirm embeddings (512D) align semantically across languages using cosine similarity.
* **Outcome**: A high-performing LLM for language-agnostic semantic search.

**Step 2: Generate Cross-Lingual Embeddings for the Knowledge Base**

* **Objective**: Embed entries for efficient retrieval using a single FAISS index, with separate embeddings for testing.
* **Action**: Generate single-vector embeddings for text and store in FAISS, while saving separate embeddings for observation, problem\_cause, and solution in a side dataset.
* **Details**:
  + Load combined\_data.csv.
  + Generate single embeddings for text using the LLM.
  + Build a single IndexFlatL2 FAISS index for dense retrieval.
  + Prepare BM25 model for sparse retrieval (tokenize text).
  + Store metadata (project, language, etc.) in a CSV for filtering.
  + Generate separate embeddings for observation, problem\_cause, solution and save to multi\_embeddings.pkl.
  + Save outputs: knowledge\_base\_index.faiss, bm25\_model.pkl, metadata.csv, multi\_embeddings.pkl, knowledge\_base\_with\_index.csv.
* **Outcome**: A searchable knowledge base with single-vector embeddings and multivector testing data.

**Step 3: Set Up Query Handling**

* **Objective**: Process queries in any of the 7 languages.
* **Action**: Detect query language and generate a multilingual embedding.
* **Details**:
  + Use FastText for rapid language detection (e.g., en, fr, it, kk, ru, es, sv).
  + Normalize and spell-check queries (lowercase, remove special characters).
  + Generate a single 512D embedding using the LLM.
* **Outcome**: A query embedding ready for cross-lingual search.

**Step 4: Perform Hybrid Similarity Search**

* **Objective**: Retrieve relevant entries with high accuracy, then filter results.
* **Action**: Combine dense (FAISS) and sparse (BM25) retrieval, followed by metadata filtering.
* **Details**:
  + Search FAISS index with query embedding for top-k results (e.g., top-50) using cosine similarity.
  + Rerank with BM25 scores to boost keyword matches.
  + Filter results by metadata (e.g., project, problemcode) post-search.
* **Outcome**: Accurate, filtered search results tailored to user criteria.

**Step 5: Translate and Cache Responses**

* **Objective**: Deliver consistent, language-appropriate responses.
* **Action**: Translate retrieved entries using MarianMT (Kazakh), caching results.
* **Details**:
  + Use DeepL API for English, French, Italian, Russian, Spanish, Swedish.
  + Use MarianMT (Hugging Face) for Kazakh with greedy decoding for consistency.
  + Cache translations in a key-value store (e.g., dictionary or Redis) linked to entry index and target language.
* **Outcome**: Fast, high-quality translations with caching for efficiency.

**Step 6: Deliver the Result**

* **Objective**: Provide clear, user-friendly answers.
* **Action**: Fetch cached translations or translate on-the-fly and cache.
* **Details**:
  + Check cache for existing translations by entry index and language.
  + If absent, translate with or MarianMT then store in cache.
  + Format responses for readability (e.g., “Observation: … Solution: …”).
* **Outcome**: Accurate responses with minimal latency.

**Step 7: Integrate with the Current System**

* **Objective**: Upgrade the existing chatbot infrastructure.
* **Action**: Integrate embeddings, hybrid search, and translation pipeline.
* **Details**:
  + Replace old FAISS indices with the new single index.
  + Update backend to handle hybrid search, filtering, and translation caching.
  + Ensure frontend sends query, project ID, and optional filters (e.g., problemcode).
* **Outcome**: A seamless, enhanced chatbot system.

**Step 8: Test and Refine**

* **Objective**: Optimize performance and evaluate multivector approach.
* **Action**: Test with real-world queries and refine the system.
* **Details**:
  + Measure retrieval accuracy (precision, recall, MRR) for single-vector pipeline.
  + Test multivector approach: Use separate embeddings from multi\_embeddings.pkl to compare retrieval performance.
  + Validate translation quality (BLEU scores, native speaker review) for DeepL and MarianMT.
  + Optimize FAISS (e.g., switch to IndexIVFFlat) and caching for speed.
* **Outcome**: A deployment-ready chatbot with insights on single vs. multivector embeddings.
* **Step 9: Handling new entries**

**Objective**: create a function that will preprocess the new entries and append it to existing knowledge base then store embedding.